Doing the impossible with FSEDIT and AF: How a screen and frame can be synchronized

Richard A. DeVenezia

Abstract

Many data entry scenarios involve two datasets related by a key. A header dataset which contains one observation and a detail dataset which contains several observations per key value. Typically an AF frame is implemented with the header information displayed in fields above an extended table which contains the detail information. But what happens when the header information contains hundreds of variables? Maintaining a frame entry with hundreds of objects can be harrowing to say the least. An FSEDIT screen is better suited for such large scale data entry; a frame can still be used to enter the detail data.

This paper describes techniques used in a SAS/FSP FSEDIT screen which invoke a SAS/AF FRAME. Some items of discussion are data structures, pmenus, command parsing, window switching using NEXT and the use of global SAS/SCL lists.

The Problem

An inspection process collects a multitude of parameters regarding some material tested. The identification, properties such as size and weight, customer test requirements and test equipment setup are just a few. Many of the default values for the equipment setup are maintained in lookup tables. Each setup is used to detect and record defects in the material.

The data collected is a crucial first step for the Quality Assurance department, allowing internal review and reporting selected information to customers.

Data Sets

Two data structures were designed and implemented as SAS datasets. The first dataset, called the *header*, records the material and the test equipment setup and has a large number of variables (>100); the second dataset called the *detail*, records the defects and has a few variables (10). Observations in the datasets are related by a key variable's value. If there are no defects in the material there are no observations in the *detail* dataset.

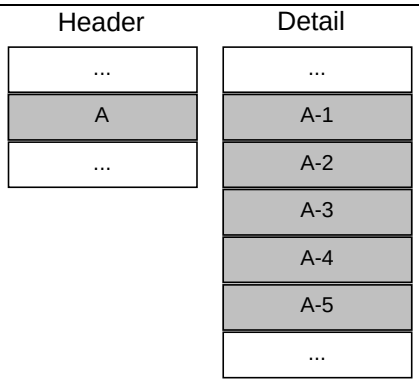
Header

Key	Variables used to relate datasets
$V_{material}$	Variables regarding material identification.
V_{test}	Variables regarding test equipment setup.
V_{spec}	Variables regarding procedures controlling setup and operation. They also describe how to classify a defect.

Detail

Key	Variables used to relate datasets
V_{defect}	Variables which record who, when, where and what kind of defect was detected. The defect is also evaluated and categorized per V_{spec} in the *header*

Dataset Relation



For each *header* observation there are zero or more *detail* observations, related by the key variable. In this case material identified with *header* key value **A** has five *details*. This is a very typical scenario which is presented using a typical frame.

Typical Frame

```
Header
Key  A_____
Vars _______ _______ _______
     _______ _______ _______
     _______ _______ _______

Detail - Extended Table
A-1 ____ ____ ____ _ ___ __
A-2 ____ ____ ____ _ ___ __
A-3 ____ ____ ____ _ ___ __
```

This type of presentation of information is suitable when only a few variables from a *header* are to be displayed.

The difficulty lies in presenting all the variables in the *header* in a consumable form and maintaining

a synchronous display of *detail.* All the variables cannot be placed on one FRAME, maintaining multiple frames to display and interact all the *header* variables is tedious and troublesome. SAS Institute has already provided an excellent tool for such problems, it is called FSEDIT.

Development Tools

An FSEDIT screen was designed to manage the *header* data and an AF FRAME was designed to manage the *detail* data. PROC FSEDIT with the *modify* option and the FSEDIT window with the *Local/Modify* menu choice were used to enter the FSEDIT development area. The BUILD command was used to start the AF development area.

FSEDIT provided multi-screen capability with numerous data subsetting, observation and variable positioning commands for advanced users. Simple pmenus were attached for the inspectors entering the data. Note: There were also several secondary tables used to populate and validate variables in *header*. None of those issues are addressed in this paper, but they are mentioned to emphasize that the basic technique demonstrated here is useable in complicated systems.

FSEDIT was excellent for rapid prototyping during development while variable grouping and coloring was changing almost daily. Also the *repeated* variable feature in modify menu item #2, Screen Modification and Field Identification, was very handy for showing the same important information on each screen.

SAS/AF provided object layout and a large code space (far more than say FSVIEW equations) and all the benefits of SCL control over data entry.

It was a challenge making these seemingly separate SAS components interact.

Application Components

Datasets
<lib>.HEADER
<lib>.DETAIL

Catalog Entries
<lib>.INSPECT.MAIN.FRAME/SCL
<lib>.INSPECT.HEADER.SCREEN
<lib>.INSPECT.DETAIL.FRAME/SCL
<lib>.INSPECT.HEADER0.PMENU
<lib>.INSPECT.HEADER1.PMENU
<lib>.INSPECT.DETAIL.PMENU

The two applications, *HEADER.screen* and *DETAIL.frame*, were developed separately with the knowledge that there would be interaction later on. They were designed to be as stand-alone as possible for testing and debugging purposes.

The *header* screen comprises four pages which are accessible via the FSEDIT commands *right, left* and =<*pageno*> and has a PMENU attached to it.

The *detail* frame has a protected object to display the key and a push-button for appending new details. For editing each *detail* an extended table named *DET_LIST* with a get row section named *GET_DET* and a put row section named *PUT_DET* was created. There is also a PMENU attached to the frame.

The *header* screen is called from a simple *MAIN* frame that has a few objects which set up macro variables. One of those objects is a check box named *doDetail,* when it is selected a macro variable *doDetail* is set to 1; when unselected 0. This macro variable determines whether the *header* screen will start the *detail* frame and indicates which PMENU to use on the screen (HEADER0 or HEADER1). A push button on the *main* frame launches the *header* screen.

HEADER Screen

Launched from the *main*:

- `call EXECCMD ('SETPMENU HEADER' || symget('doDetail') || ';');`
- `call FSEDIT ('<lib>.HEADER', '<lib>.INSPECT.HEADER');`

The first statement selects which pmenu is attached to the *header* screen which is started with the second statement. See end of paper for PMENU source code.

Several important statements are issued in the **FSEINIT** section:

- `call WNAME ('HEADER');`
- ```
  depth_gain = makelist (0, 'G');
  genv = envlist ('G');
  genv = setniteml (genv, depth_gain,
  'INSPECTION HEADER: DEPTH_GAIN');
  ```
- ```
  rej_criteria = makelist (0, 'G');
  genv = envlist ('G');
  genv = setniteml (genv, rej_criteria,
  'INSPECTION HEADER: REJ_CRITERIA');
  ```
- ```
  if symgetN ('doDetail') then
     call EXECCMD ('AFA
  C=<lib>.INSPECT.DETAIL.FRAME AWS=NO;');
  ```

The WNAME statement assures we have a known window name. There are NEXT commands issued via *detail* PMENU selections which rely on the window name. The makelist statements create two empty lists which are then pointed to from the
  ```

global environment using the setniteml statement. These lists are used later to communicate parameters for evaluating and categorizing defects. The EXECCMD statement launches the *detail* frame if requested in the *main.*

The **INIT** section is the all important place where the *detail* frame becomes synchronized with the *header* screen. First there are links to labeled sections which load the SCL lists *depth_gain* and *rej_criteria*. Also, any edits which cause changes in variables regarding depth_gain or rej_criteria will eventually link to an appropriate labeled section in the SCL to update the SCL lists. What values are placed in the SCL lists is not important to this paper, only the fact that some content is put there and will need to be retrieved in the *detail* frame.

The *detail* frame is then given the key value to display:

```
• if symgetN ('doDetail') then
     call EXECCMD ('NEXT DETAIL;'
                || 'SETKEY ' ||
  quote(key)
                || ';NEXT HEADER;');
```

The rather complicated EXECCMD switches control to the *detail* frame, and issues a custom command SETKEY, which is parsed by the frame in it's main section (The custom command does not cause an error because the frame issued a 'control always' in it's INIT section). Control then returns to the *header* screen.

The SETKEY command sent to the DETAIL frame, via EXECCMD, looks like the following:

```
SETKEY "XYZZY A"
```

The key value part of the command is enclosed in quotes because the key value may have embedded spaces. The WORD function used in the DETAIL frame to parse the command line will see the command as two words, making it easy to extract the key value. The key value is then used in a where statement.

The **FSETERM** section contains statements to delete SCL lists and end the *detail* frame:

```
• depth_gain = dellist (depth_gain);
  genv = envlist ('G');
  genv = delniteml (genv, 'INSPECTION
  HEADER: DEPTH_GAIN');
• rej_criteria=dellist(rej_criteria);
  genv = envlist ('G');
  genv = delniteml (genv, 'INSPECTION
  HEADER: REJ_CRITERIA');
• if symgetN ('doDetail') then
     call EXECCMDI ('NEXT DETAIL; ENDNOW;
  END; NEXT HEADER;');
```

The EXECCMDI statement switches to the *detail* frame, issues a custom command ENDNOW, issues an END and returns control to the *header* screen.

DETAIL Frame

Like the FSEINIT section of the *header* screen the **INIT** section of the *detail* frame names the window:

```
• call WNAME ('DETAIL');
• _detail = open ('<lib>.DETAIL','u');
• call SET (_detail);
• control always;
• noEnd = 1;
```

The *detail* dataset must be opened explicitly in the frame application. The *header* dataset is opened automatically when the *main* frame issues the 'call FSEDIT'. The call SET statement causes automatic data value transfer between the dataset data vector and the SCL data vector. The control statement allows the frame to run without an error when a custom command is issued.

Since the *detail* frame is launched from the *header,* it is appropriate that the frame cannot close with the just an END or CANCEL command. A custom command ENDNOW must be issued first. (Note: The ENDNOW command is issued in the FSETERM section of the *header* screen.) The noEnd variable is examined in the TERM section of the frame, control is returned to the frame if noEnd=1.

The **MAIN** section processes the custom commands issued by the *header* screen or the *detail* PMENU:

```
• select (word(1, 'U'));
    when ('SETKEY') do;
      call nextword();
      rc=where(_detail,'key='||quote(word(1)));
      call notify('DET_LIST','_NEED_REFRESH_');
                    end;
    when ('ENDNOW') noEnd = 0
    when ('DELETE') link DELETE;
  end;
• call nextword();
```

When the SETKEY command is issued, the where on the *detail* data is reapplied using the specified key value, and the extended table

DET_LIST is instructed to redisplay itself, causing the observations of the new key to appear.

When the ENDNOW command is issued the flag variable controlling the TERM section, *noEnd*, is set to 0.

When the DELETE command is issued a labeled section is run. The details of the delete will not be discussed.

The **TERM** section closes the *detail* dataset if allowed:

```
• if noEnd then do;
    _status_ = 'R';
    return;
  end;
• _detail = close (_detail);
```

If noEnd is 1 then control is returned to the frame by setting the _status_ variable to R.

The get row section **GET_DET** reads observations from the *detail* dataset and populates the extended table automatically due to the earlier call SET:

```
• if fetchobs (_detail, _CURROW_) then
    call notify
  ('DET_LIST','_ENDTABLE_');
```

The put row section **PUT_DET** updates observations when a row in the extended table has had one or more objects modified:

```
• rc =
  fetchobs(_detail,_CURROW_,'NOSET');
• rc = update  (_detail);
```

The fetchobs uses the NOSET option so the newly modified field values do not get overwritten automatically by the fetch, due to earlier call SET. The fetchobs essentially positions a pointer to the appropriate observation and the update writes the current variable values to the observation.

The global SCL lists pointed to by the global environment list items named **INSPECTION HEADER: DEPTH_GAIN** and **INSPECTION HEADER: REJ_CRITERIA** are used in object named sections. Two of the several objects in the extended table row are named *depth* and *eval*.

```
• depth:
    genv = envlist ('G');
```

The Implementation

```
    depth_gain = getniteml (genv,
  'INSPECTION HEADER: DEPTH_GAIN');
    ...
  return;
• eval:
    genv = envlist ('G');
    rej_criteria = getniteml (genv,
  'INSPECTION HEADER: REJ_CRITERIA');
    ...
  return;
```

In both cases, a global SCL list populated in the *header* screen is referenced, by obtaining a pointer to the SCL list from the global environment list using getniteml. Each labeled section further uses the SCL list obtained to set another variables value according to an inspection procedures rules. Also, since an object in an extended table row was modified, control will pass to the **PUT_DET** section where the new variable values are written to the *detail* dataset.

Conclusion

SAS offers two application development tools SAS/FSP FSEDIT and SAS/AF FRAME which are able to interact using a combination of display manager commands and custom commands. Furthermore, application specific information can be passed between components using global SCL lists.

Author

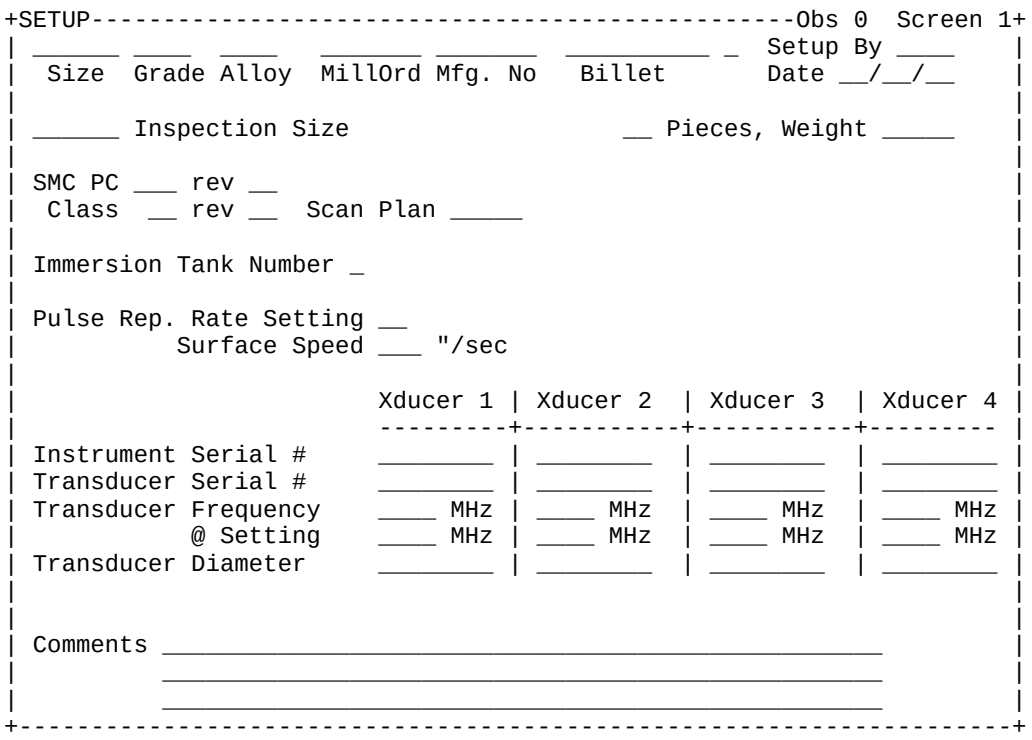
Richard DeVenezia
9949 East Steuben Road
Remsen NY 13438.
(315) 831-4101.
radevenz@ix.netcom.com

A SAS Institute Quality Partner with SGS Statistical Services.

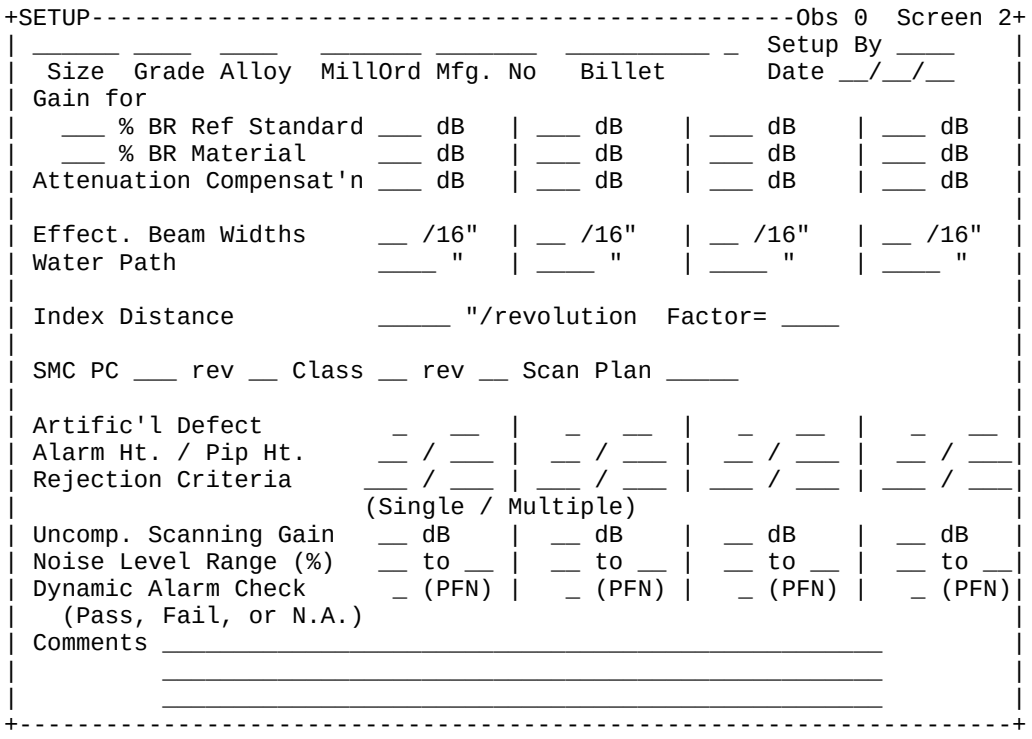
References

At submittal time, after being in operation for approximately six months, 2007 setups and 995 defects had been recorded. The *header* (setups) has 148 variables and an observation length of 625 bytes. The *detail* (defects) has 13 variables and an observation length of 59 bytes. At this time the datasets are not compressed, and are in libraries managed by SAS/SHARE to allow concurrent access to the data.

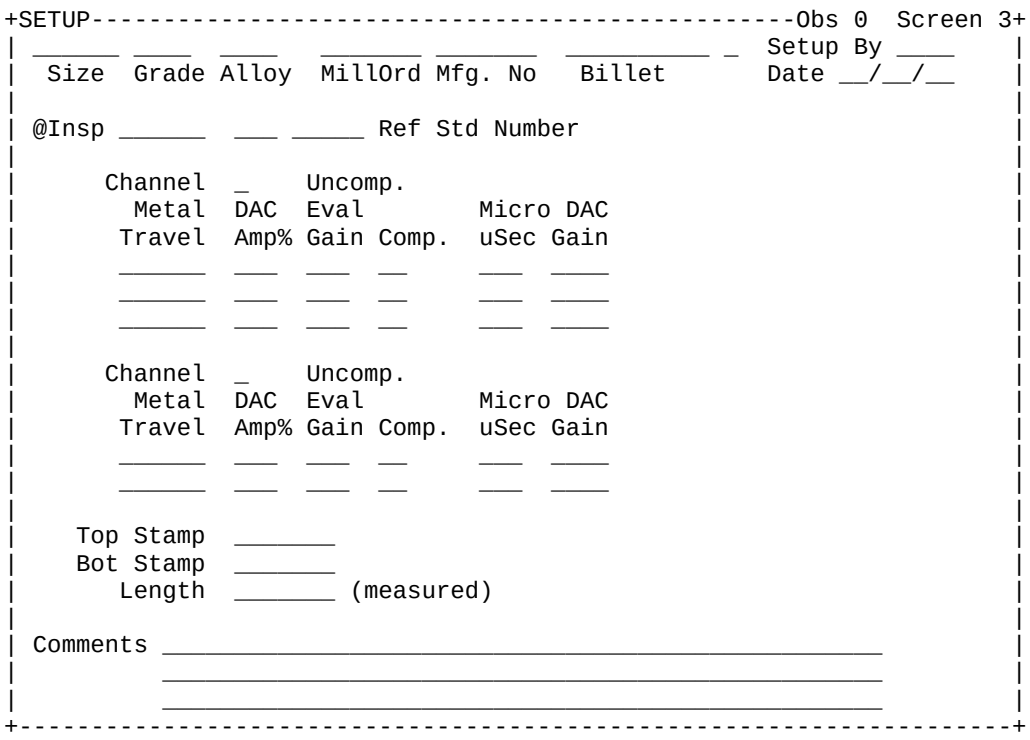
Header Screen #1



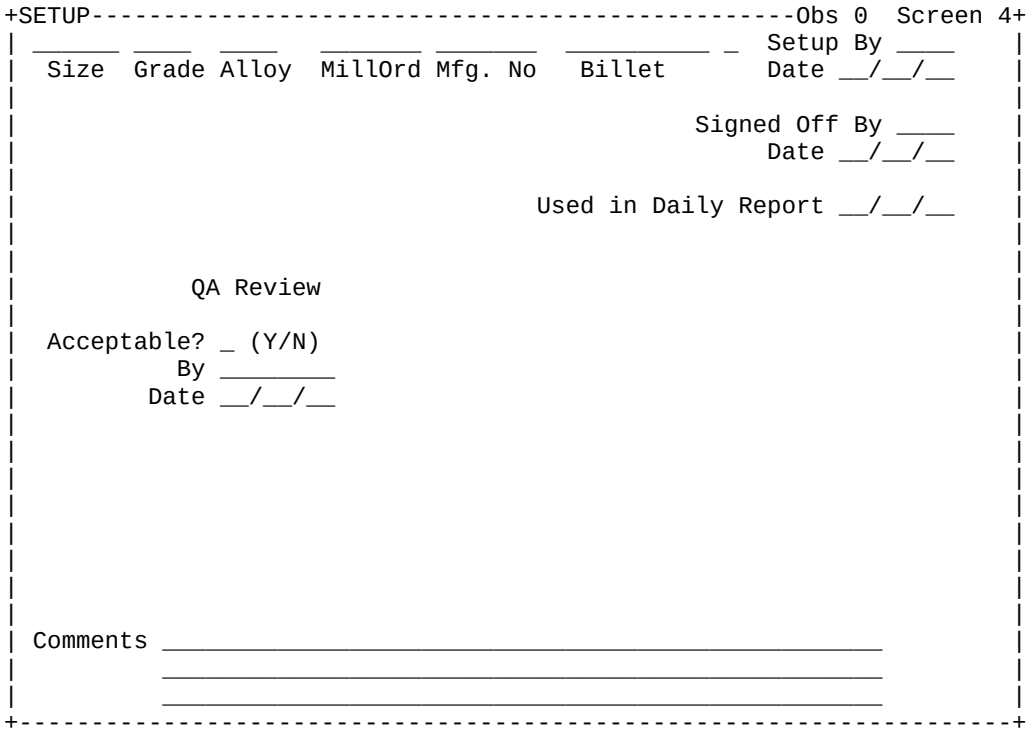
Header Screen #2



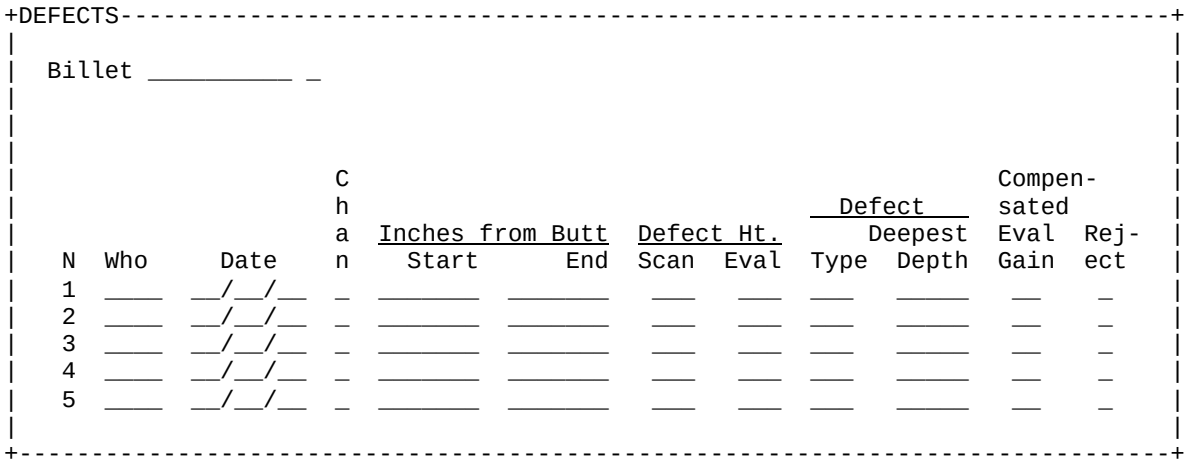
Header Screen #3



Header Screen #4



Detail Frame



PMENU appearance

Header.screen

File
- Print Screen
- ----------
- End

View
- Screen 1 - Transducers
- Screen 2 - Parameters
- Screen 3 - Reference
- Screen 4 - QA Review
- ----------
- Details

Detail.frame

File
- Print Screen
- ----------
- End

View
- Screen 1 - Transducers
- Screen 2 - Parameters
- Screen 3 - Reference
- Screen 4 - QA Review

Special
- Delete a detail observation

PMENU commands

The commands issued by PMENU selections are an important part of the application. They issue the FSEDIT command ***=<pageno>*** and the Display Manager command ***NEXT***.

```
proc pmenu cat=<lib>.inspect;

  menu HEADER0;     * Does not have details option;

    item 'File' menu=file;
    item 'View' menu=view;
  menu file;
    item 'Print Screen' mnemonic='P' selection=PRINT;
    separator;
    item 'End'          mnemonic='E';
    selection PRINT  'SPRINT';
  menu view;
    item 'Screen 1 - Transducers' mnemonic='1' selection=_1;
    item 'Screen 2 - Parameters'  mnemonic='2' selection=_2;
    item 'Screen 3 - Reference'   mnemonic='3' selection=_3;
    item 'Screen 4 - QA Review'   mnemonic='4' selection=_4;
    selection _1 '=1';
    selection _2 '=2';
    selection _3 '=3';
    selection _4 '=4';
  run;


  menu HEADER1;      Has detail option;

    item 'File' menu=file;
    item 'View' menu=view;
  menu file;
    item 'Print Screen' mnemonic='P' selection=PRINT;
    separator;
    item 'End'          mnemonic='E';
    selection PRINT  'SPRINT';
  menu view;
    item 'Screen 1 - Transducers' mnemonic='1' selection=_1;
    item 'Screen 2 - Parameters'  mnemonic='2' selection=_2;
    item 'Screen 3 - Reference'   mnemonic='3' selection=_3;
    item 'Screen 4 - QA Review'   mnemonic='4' selection=_4;
    separator;
    item 'Details'  mnemonic='D' selection=detail;
    selection _1 '=1';
    selection _2 '=2';
    selection _3 '=3';
    selection _4 '=4';
    selection detail 'NEXT DETAIL';
  run;


  menu DETAIL;

    item 'File'    menu=file;
    item 'View'    menu=view;
    item 'Special' menu=special;
  menu file;
    item 'Print Screen' mnemonic='P' selection=PRINT;
    separator;
    item 'End'          mnemonic='E' selection=END;
    selection PRINT  'SPRINT;';
    selection END    'NEXT HEADER;END;';
  menu view;
    item 'Header Screen 1 - Transducers' mnemonic='1' selection=_1;
    item 'Header Screen 2 - Parameters'  mnemonic='2' selection=_2;
    item 'Header Screen 3 - Reference'   mnemonic='3' selection=_3;
    item 'Header Screen 4 - QA Review'   mnemonic='4' selection=_4;
    selection _1 'NEXT HEADER;=1;';
    selection _2 'NEXT HEADER;=2;';
    selection _3 'NEXT HEADER;=3;';
    selection _4 'NEXT HEADER;=4;';
  menu special;
    item 'Delete a detail observation' mnemonic='D' selection=delete;
    selection delete 'DELETE';
  run;
quit;
```